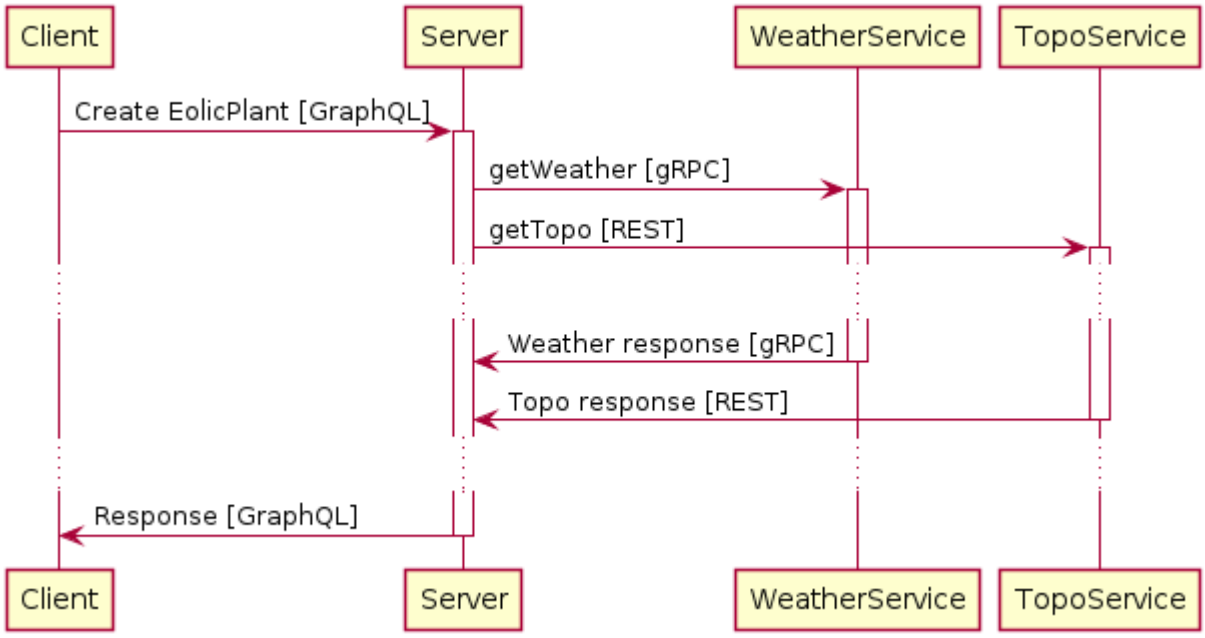
Práctica 3. Protocolos y reactividad

Enunciado

Se pide crear una aplicación distribuida para la generación de plantas eólicas: EoloPlanner. Esta aplicación está compuesta por los siguientes servicios:

- Client: Código JavaScript ejecutado en un navegador web.
- Server: Servidor web que expone una API GraphQL que será usada por el cliente.
- WeatherService: Devuelve información meteorológica.
- TopoService: Devuelve información sobre la orografía del terreno.

Estos servicios se comunican de la siguiente forma:



A continuación se detalla esta comunicación:

- Client usa la API GraphQL para enviar una petición de creación de una Planta Eólica (Create EolicPlant).
- Cuando el Server recibe la petición de creación de una nueva planta eólica del Client, solicita la información meteorológica y topográfica a los servicios WeatherService y TopoService respectivamente.
- El servicio WeatherService ofrecer una interfaz gRPC.
- El servicio TopoService ofrece una interfaz REST.

Algunos detalles de implementación:

- **Client**
  - Se servirá por http como estático en el servidor web del Server.
  - Se implementará como AJAX: Código JavaScript llamando a la API GraphQL. Se puede asumir que el browser es moderno y tiene soporte JavaScript y estándares modernos. Se puede usar alguna librería externa si se considera útil.
  - La web del cliente mostrará las plantas del server.
  - También proporcionará un formulario para incluir una ciudad y un botón para crear una planta en esa ciudad. Cuando se pulse el botón el botón de crear se realizará una petición GraphQL al Server para solicitar la creación de la planta.
  - La interfaz sólo permitirá crear una creación de una planta eólica a la vez. Para ello deshabilitará el botón hasta que se haya creado y mostrará un spinner.
  - La web estará disponible en la ruta raíz del servidor web http://127.0.0.1:3000/

- **Server**
  - Ofrecerá una API GraphQL para crear, listar y borrar plantas eólicas.
  - Se implementará con Node.js y con una base de datos MySQL.
  - Una planta se crea con el nombre de una ciudad. El resultado de la creación de una planta eólica será una planificación en forma de texto.
    - Creación
      - Datos de entrada: `{ "city": "Madrid" }`
      - Respuesta: `{ "id": 1, "city": "Madrid", "planning": "madrid-sunny-flat" }`
    - Consulta
      - Respuesta: `{ "id": 1, "city": "Madrid", "planning": "madrid-sunny-flat" }`
    - El servidor deberá soportar la creación de varias plantas en paralelo solicitadas por varios usuarios (se podrá simular desde varias pestañas del navegador). No deberá haber interferencias entre ellas.
  - Cuando tenga que realizar una nueva planta eólica llamará a los servicios WeatherService y TopoService en paralelo.
  - Se simulará un tiempo de proceso de 1 a 3 segundos aleatorio.
  - El resultado de la creación de la planta se creará de la siguiente forma: la ciudad concatenada a la respuesta del servicio que responda primero concatenada a la respuesta del segundo. El resultado se convertirá a lowercase si la ciudad empieza por una letra igual o anterior a M o en uppercase si es posterior.

- **WeatherService**
  - Ofrecerá una API gRPC en la que recibirá una ciudad como parámetro y devolverá Rainy o Sunny.
  - Se implementará en Node.
  - Para simular el cálculo interno devolverá “Rainy” si la ciudad empieza por vocal y “Sunny” si empieza por consonante.
  - Simulará un tiempo de proceso de 1 a 3 segundos aleatorio con un setTimeout.

- El formato de la API gRPC será:

```
service WeatherService {
   rpc GetWeather(GetWeatherRequest) returns (Weather);
}

message GetWeatherRequest {
   string city = 1;
}

message Weather {
   string city = 1;
   string weather = 2;
}
```

- **TopoService**
  - Ofrecerá una API REST en la que recibirá una ciudad como parámetro y devolverá “Flat” o “Mountain”.
    - Consulta
      - URL: `/api/topographicdetails/Madrid`
      - Method: GET
      - Response Body: `{ "id": "Madrid", "landscape": "flat" }`
  - Se implementará en Java con SpringBoot.
  - Tendrá una lista de ciudades asociadas a su landscape “Flat” o “Mountain” guardadas en una MongoDB.
  - Se implementará de forma reactiva funcional con WebFlux.
  - Simulará un tiempo de proceso de 1 a 3 segundos aleatorio.

Otras consideraciones:

- Se asume que las bases de datos (MySQL y MongoDB) están disponibles en los puertos por defecto en localhost.
- Se prestará especial cuidado para que la lógica de negocio (aunque esté muy simplificada) esté lo más desacoplada posible de las librerías utilizadas para la comunicación entre servicios.

El proyecto se deberá estructurar de la siguiente forma:

- Habrá una carpeta por cada módulo: server, weatherservice y toposervice.
- El client estará alojado como un estático en el server.
- Al entregar la práctica no se deben incluir las carpetas node_modules de los proyectos Node ni la carpeta target de los proyectos Maven.
- Para facilitar la construcción de la aplicación distribuida se incluirá el siguiente script Node (install.js) en la raíz que permitirá la instalación de las dependencias de todos los proyectos. Este script se ejecutará en la shell como: ‘node install.js’:

```
const { spawnSync } = require('child_process');

function exec(serviceName, command){

  console.log(`Installing dependencies for [${serviceName}]`);
  console.log(`Folder: ${serviceName} Command: ${command}`);

  spawnSync(command, [], {
    cwd: serviceName,
    shell: true,
    stdio: 'inherit'
  });
}

exec('weatherservice', 'npm install');
exec('toposervice', 'mvn install');
exec('server','npm install');
```

- Para facilitar la ejecución de la aplicación distribuida se incluirá el siguiente script Node (exec.js) en la raíz que permitirá la ejecución de los diferentes servicios. Este script se ejecutará como: 'node exec.js'.

```
const {spawn} = require('child_process');

function exec(serviceName, command) {

console.log(`Started service [${serviceName}]`);

  let cmd = spawn(command, [], {cwd: './' + serviceName, shell: true});

  cmd.stdout.on('data', function (data) {
    process.stdout.write(`[${serviceName}] ${data}`);
  });

  cmd.stderr.on('data', function (data) {
    process.stderr.write(`[${serviceName}] ${data}`);
  });

  return cmd;
}

const services = new Map();

services.set('weatherservice', exec('weatherservice', 'node src/server.js'));
services.set('toposervice', exec('toposervice', 'mvn spring-boot:run'));
services.set('server', exec('server', 'node src/server.js'));

process.on('SIGINT', async () => {
  for (var [name, cmd] of services) {
    console.log(`Killing service [${name}]`);
    cmd.stdin.pause();
    await cmd.kill();
  }
  process.exit();
});
```

- Las bases de datos externas pueden ejecutarse con docker usando el siguiente script Node (exec_aux_services.js):

```
const { spawnSync } = require('child_process');

function exec(serviceName, command){

  console.log(`Starting docker image for [${serviceName}]`);
  console.log(`Command: ${command}`);

  spawnSync(command, [], {

    shell: true,
    stdio: 'inherit'
  });
}

exec('MongoDB', 'docker run --rm -d -p 27017-27019:27017-27019 --name mongodb mongo');
exec('MySQL', 'docker run --rm -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=eoloplantsDB -p
3306:3306 --name mysql mysql:8.0.22');
```

Formato de entrega

La práctica se entregará teniendo en cuenta los siguientes aspectos:

- La práctica se entregará como un fichero .zip. El nombre del fichero .zip será el correo URJC del alumno (sin @alumnos.urjc.es).
- En la raíz del fichero .zip debe existir un fichero README.md que explique cómo ejecutar los cuatro módulos de la aplicación partiendo del código fuente del .zip.
- Los proyectos se pueden crear con cualquier editor o IDE (para el proyecto Node se recomienda VSCode).
- La práctica se entregará por el aula virtual con la fecha indicada.
- No se deberán incluir en el .zip las carpetas node_modules ni target ya que contienen dependencias o código compilado que se puede regenerar partiendo de los fuentes.

Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas:

- Sólo será entregada por uno de los alumnos
- El nombre del fichero .zip contendrá el correo de ambos alumnos separado por guión. Por ejemplo `p.perezf2019-z.gonzalez2019.zip`